**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 3

**З дисципліни:** *“Кросплатформне програмування”*

**На тему:** *“Робота з текстовою інформацією, файлами та серіалізація”*

**Лектор:**

доц. каф. ПЗ

Дяконюк Л. М.

**Виконав:**

ст. гр. ПЗ-35

Хруставчук М.Л.

**Прийняв:**

ст. викл. каф. ПЗ

Шкраб Р. Р.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2024 р.

∑= \_\_\_\_\_ .

Львів – 2024

**Тема роботи:** Робота з текстовою інформацією, файлами та серіалізація.

**Мета роботи:** Навчитися працювати з потоками введення-виведення (I/O Streams), текстовою інформацією, файлами, нативною серіалізацією та сторонніми бібліотеками для серіалізації об'єктів у формати JSON та YAML. Ознайомитися з використанням Maven для управління залежностями та збіркою проекту, вивчити особливості використання систем збірки проєктів.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

1. I/O Streams (Вводу-Виводу потоки):

* Потоки вводу-виводу (I/O Streams) в Java використовуються для читання та запису даних з різних джерел, таких як файли, мережа або системні ресурси. Існують буферизовані потоки, які зменшують кількість фізичних операцій читання/запису, та небуферизовані, які працюють напряму з джерелом даних.

1. Maven:
   * + Maven – це популярна система автоматизованої збірки проектів для Java, яка допомагає керувати залежностями, збирати проекти та організовувати їх структуру. Вона використовує файл pom.xml, у якому зазначаються залежності та налаштування проекту.
2. JSON (JavaScript Object Notation):

* JSON – це легкий формат обміну даними, який зручно використовувати для передачі структурованих даних між сервером і клієнтом. Він широко застосовується в веб-розробці для серіалізації об'єктів.

1. YAML (YAML Ain't Markup Language):

* YAML – це простий у використанні формат серіалізації даних, що підтримує читання та редагування людиною. Він часто використовується для конфігураційних файлів завдяки своїй читабельності та гнучкості.

1. CSV (Comma-Separated Values):
   * + CSV – це простий формат для зберігання табличних даних, де кожен рядок є записом, а стовпці відокремлені комами. CSV широко використовується для імпорту та експорту даних у базах даних і таблицях.
2. Maven

* Maven – система автоматизованої збірки, яка допомагає керувати проектом та його залежностями через XML конфігурації. Gradle – більш сучасна система збірки, яка використовує DSL (Domain Specific Language) для налаштування, дозволяючи більш гнучку конфігурацію проектів.

**ЗАВДАННЯ**

*Варіант 3. Бібліотечна система*

1. Робота з потоками вводу-виводу:
   * Читання та запис інформації про книги (назва, автор, рік видання) у файл.
   * Використайте буферизовані потоки для збереження назв книг та небуферизовані для збереження авторів.
   * Умова: Поле рік видання книги не повинно бути серіалізоване.
2. Нативна серіалізація Java:
   * Серіалізуйте об'єкти книг у файл.
   * Умова: Книги без автора не повинні серіалізуватися.
3. Серіалізація JSON (GSON або Jackson):
   * Серіалізуйте книги у JSON, пропускаючи поле рік видання.
   * При читанні відновіть усі інші поля.
4. Серіалізація YAML (SnakeYAML):
   * Серіалізуйте книги у формат YAML.

Умова: Поле автор має бути серіалізоване лише для книг виданих до 2000 року.

**ХІД ВИКОНАННЯ**

**1. Код розробленої програми**

Main.java

package org.example;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 char answer;  
 System.*out*.print("Введіть номер завдання, яке треба виконати (1-4): ");  
 answer = scanner.next().charAt(0);  
  
 if (answer == '1') {  
 *IOSerialization*();  
 } else if (answer == '2') {  
 *NativeSerialization*();  
 } else if (answer == '3') {  
 *JsonSerialization*();  
 } else if (answer == '4') {  
 *YamlSerialization*();  
 } else {  
 System.*out*.println("Некоректне введення. Спробуйте ще раз.");  
 }  
 }  
  
 public static void IOSerialization() {  
 Author author1 = new Author("Іван", "Франко");  
 Author author2 = new Author("Леся", "Українка");  
  
 Book book1 = new Book("Захар Беркут", author1, 320, 1883);  
 Book book2 = new Book("Лісова пісня", author2, 200, 1911);  
  
 List<Book> books = new ArrayList<>();  
 books.add(book1);  
 books.add(book2);  
  
 String fileName = "books\_io.txt";  
  
 for (Book book : books) {  
 BookIO.*writeBook*(fileName, book);  
 }  
  
 List<Book> readBooks = BookIO.*readBooks*(fileName);  
 for (Book book : readBooks) {  
 System.*out*.println("Книга: " + book.getTitle() + ", Автор: " + book.getAuthor().getName() +  
 " " + book.getAuthor().getSurname());  
 }  
 }  
  
 public static void NativeSerialization() {  
 Author author1 = new Author("Іван", "Франко");  
 Author author2 = new Author("Леся", "Українка");  
  
 Book book1 = new Book("Захар Беркут", author1, 320, 1883);  
 Book book2 = new Book("Лісова пісня", author2, 200, 1911);  
 Book book3 = new Book("Книга без автора", null, 150, 2000);  
  
 List<Book> books = new ArrayList<>();  
 books.add(book1);  
 books.add(book2);  
 books.add(book3);  
  
 String fileName = "books.ser";  
  
 NativeBookSerializer.*serializeBooks*(books, fileName);  
  
 List<Book> deserializedBooks = NativeBookSerializer.*deserializeBooks*(fileName);  
  
 for (Book book : deserializedBooks) {  
 System.*out*.println("Книга: " + book.getTitle() + ", Автор: " + book.getAuthor().getName() +  
 " " + book.getAuthor().getSurname() + ", Сторінки: " + book.getPages());  
 }  
 }  
  
 public static void JsonSerialization() {  
 Author author1 = new Author("Іван", "Франко");  
 Author author2 = new Author("Леся", "Українка");  
  
 Book book1 = new Book("Захар Беркут", author1, 320, 1883);  
 Book book2 = new Book("Лісова пісня", author2, 200, 1911);  
  
 List<Book> books = new ArrayList<>();  
 books.add(book1);  
 books.add(book2);  
  
 String fileName = "books.json";  
  
 JacksonBookSerializer.*serializeBooksToJson*(books, fileName);  
  
 List<Book> deserializedBooks = JacksonBookSerializer.*deserializeBooksFromJson*(fileName);  
  
 for (Book book : deserializedBooks) {  
 System.*out*.println("Книга: " + book.getTitle() + ", Автор: " + book.getAuthor().getName() +  
 " " + book.getAuthor().getSurname());  
 }  
 }  
  
 public static void YamlSerialization() {  
 Author author1 = new Author("Іван", "Франко");  
 Author author2 = new Author("Леся", "Українка");  
  
 Book book1 = new Book("Захар Беркут", author1, 320, 1883);  
 Book book2 = new Book("Лісова пісня", author2, 200, 1911);  
 Book book3 = new Book("Книга сучасності", null, 150, 2021);  
  
 List<Book> books = new ArrayList<>();  
 books.add(book1);  
 books.add(book2);  
 books.add(book3);  
  
 String fileName = "books.yaml";  
  
 YamlBookSerializer.*serializeBooksToYaml*(books, fileName);  
  
 List<Book> deserializedBooks = YamlBookSerializer.*deserializeBooksFromYaml*(fileName);  
  
 for (Book book : deserializedBooks) {  
 String authorInfo = (book.getAuthor() != null) ?  
 book.getAuthor().getName() + " " + book.getAuthor().getSurname() : "Невідомий автор";  
 System.*out*.println("Книга: " + book.getTitle() + ", Автор: " + authorInfo);  
 }  
 }  
}

Book.java

package org.example;  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import java.io.Serializable;  
  
public class Book implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 private String title;  
 private Author author;  
 private int pages;  
  
 @JsonIgnore  
 private transient int year;  
  
 public Book(){}  
 public Book(String title, Author author, int pages, int year) {  
 this.title = title;  
 this.author = author;  
 this.pages = pages;  
 this.year = year;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public Author getAuthor() {  
 return author;  
 }  
  
 public int getPages() {  
 return pages;  
 }  
  
 public int getYear() {  
 return year;  
 }  
}

BookIO.java

package org.example;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
import java.nio.charset.StandardCharsets;  
  
public class BookIO {  
  
 public static void writeBook(String fileName, Book book) {  
 try (BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(fileName, true))) {  
 bos.write(("TITLE:" + book.getTitle() + "\n").getBytes(StandardCharsets.*UTF\_8*));  
 bos.write(("AUTHOR:" + book.getAuthor().getName() + "," + book.getAuthor().getSurname() + "\n").getBytes(StandardCharsets.*UTF\_8*));  
 bos.write(("PAGES:" + book.getPages() + "\n").getBytes(StandardCharsets.*UTF\_8*));  
 bos.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static List<Book> readBooks(String fileName) {  
 List<Book> books = new ArrayList<>();  
 try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(fileName))) {  
 StringBuilder content = new StringBuilder();  
 byte[] buffer = new byte[1024];  
 int bytesRead;  
  
 while ((bytesRead = bis.read(buffer)) != -1) {  
 content.append(new String(buffer, 0, bytesRead, StandardCharsets.*UTF\_8*));  
 }  
  
 String[] lines = content.toString().split("\n");  
  
 for (int i = 0; i < lines.length; i += 3) {  
 String title = lines[i].replace("TITLE:", "").trim();  
 String[] authorParts = lines[i + 1].replace("AUTHOR:", "").trim().split(",");  
 int pages = Integer.*parseInt*(lines[i + 2].replace("PAGES:", "").trim());  
  
 Author author = new Author(authorParts[0], authorParts[1]);  
 books.add(new Book(title, author, pages, 0));  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return books;  
 }  
}

JacksonBookSerializer.java

package org.example;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;  
  
import java.io.File;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
  
public class JacksonBookSerializer {  
  
 private static final ObjectMapper objectMapper = new ObjectMapper().registerModule(new JavaTimeModule());  
  
 // Серіалізація списку книг у JSON-файл  
 public static void serializeBooksToJson(List<Book> books, String fileName) {  
 try {  
 objectMapper.writeValue(new File(fileName), books);  
 System.out.println("Список книг серіалізовано у JSON-файл: " + fileName);  
 } catch (IOException e) {  
 System.err.println("Помилка під час серіалізації у JSON: " + e.getMessage());  
 }  
 }  
  
 // Десеріалізація списку книг із JSON-файлу  
 public static List<Book> deserializeBooksFromJson(String fileName) {  
 try {  
 List<Book> books = objectMapper.readValue(new File(fileName),  
 objectMapper.getTypeFactory().constructCollectionType(List.class, Book.class));  
 System.out.println("Список книг десеріалізовано з JSON-файлу: " + fileName);  
 return books;  
 } catch (IOException e) {  
 System.err.println("Помилка під час десеріалізації з JSON: " + e.getMessage());  
 return new ArrayList<>();  
 }  
 }  
}

NativeBookSerializer.java

package org.example;  
  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class NativeBookSerializer {  
  
 // Серіалізація списку книг у файл  
 public static void serializeBooks(List<Book> books, String fileName) {  
 // Фільтруємо книги з порожнім автором  
 List<Book> validBooks = new ArrayList<>();  
 for (Book book : books) {  
 if (book.getAuthor() != null) {  
 validBooks.add(book);  
 } else {  
 System.*out*.println("Книга \"" + book.getTitle() + "\" не має автора, тому не буде серіалізована.");  
 }  
 }  
  
 try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName))) {  
 oos.writeObject(validBooks);  
 System.*out*.println("Список книг успішно серіалізовано у файл: " + fileName);  
 } catch (IOException e) {  
 System.*err*.println("Помилка під час серіалізації книг: " + e.getMessage());  
 }  
 }  
  
 // Десеріалізація списку книг із файлу  
 @SuppressWarnings("unchecked")  
 public static List<Book> deserializeBooks(String fileName) {  
 try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {  
 List<Book> books = (List<Book>) ois.readObject();  
 System.*out*.println("Список книг успішно десеріалізовано з файлу: " + fileName);  
 return books;  
 } catch (IOException | ClassNotFoundException e) {  
 System.*err*.println("Помилка під час десеріалізації книг: " + e.getMessage());  
 return new ArrayList<>();  
 }  
 }  
}  
YamlBookSerializer.java

package org.example;  
  
import org.yaml.snakeyaml.DumperOptions;  
import org.yaml.snakeyaml.Yaml;  
import org.yaml.snakeyaml.DumperOptions.FlowStyle;  
  
import java.io.FileWriter;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class YamlBookSerializer {  
  
 private static final Yaml yaml;  
  
 static {  
 DumperOptions options = new DumperOptions();  
 options.setDefaultFlowStyle(FlowStyle.BLOCK);  
 yaml = new Yaml(options);  
 }  
  
 // Серіалізація списку книг у YAML-файл  
 public static void serializeBooksToYaml(List<Book> books, String fileName) {  
 List<Map<String, Object>> serializedBooks = new ArrayList<>();  
  
 for (Book book : books) {  
 Map<String, Object> bookData = new HashMap<>();  
 bookData.put("title", book.getTitle());  
 bookData.put("pages", book.getPages());  
 bookData.put("year", book.getYear());  
  
 // Додаємо автора лише для книг із роком видання < 2000  
 if (book.getYear() < 2000 && book.getAuthor() != null) {  
 Map<String, Object> authorData = new HashMap<>();  
 authorData.put("name", book.getAuthor().getName());  
 authorData.put("surname", book.getAuthor().getSurname());  
 bookData.put("author", authorData);  
 }  
  
 serializedBooks.add(bookData);  
 }  
  
 try (FileWriter writer = new FileWriter(fileName)) {  
 yaml.dump(serializedBooks, writer);  
 System.out.println("Список книг серіалізовано у YAML-файл: " + fileName);  
 } catch (IOException e) {  
 System.err.println("Помилка під час серіалізації у YAML: " + e.getMessage());  
 }  
 }  
  
 // Десеріалізація списку книг із YAML-файлу  
 @SuppressWarnings("unchecked")  
 public static List<Book> deserializeBooksFromYaml(String fileName) {  
 try (FileReader reader = new FileReader(fileName)) {  
 List<Map<String, Object>> serializedBooks = yaml.load(reader);  
 List<Book> books = new ArrayList<>();  
  
 for (Map<String, Object> bookData : serializedBooks) {  
 String title = (String) bookData.get("title");  
 int pages = (int) bookData.get("pages");  
 int year = (int) bookData.get("year");  
  
 Author author = null;  
 if (bookData.containsKey("author")) {  
 Map<String, Object> authorData = (Map<String, Object>) bookData.get("author");  
 String name = (String) authorData.get("name");  
 String surname = (String) authorData.get("surname");  
 author = new Author(name, surname);  
 }  
  
 books.add(new Book(title, author, pages, year));  
 }  
  
 System.out.println("Список книг десеріалізовано з YAML-файлу: " + fileName);  
 return books;  
 } catch (IOException e) {  
 System.err.println("Помилка під час десеріалізації з YAML: " + e.getMessage());  
 return new ArrayList<>();  
 }  
 }  
}

**2. Результати виконання розробленої програми**

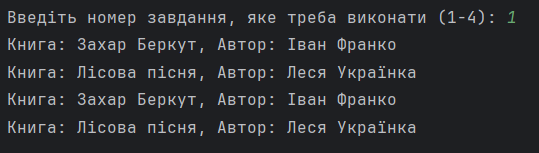


Рис. 1. Виконання IOSerialization

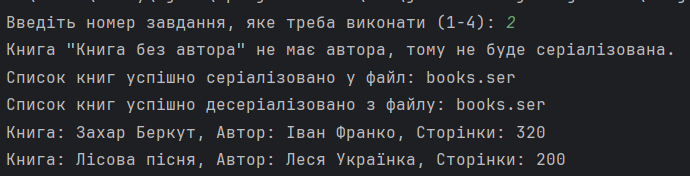


Рис. 2. Виконання IOSerialization

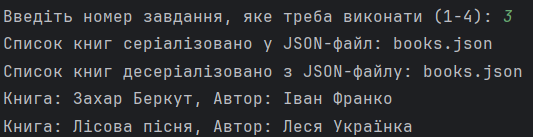


Рис. 3. Виконання JsonSerialization

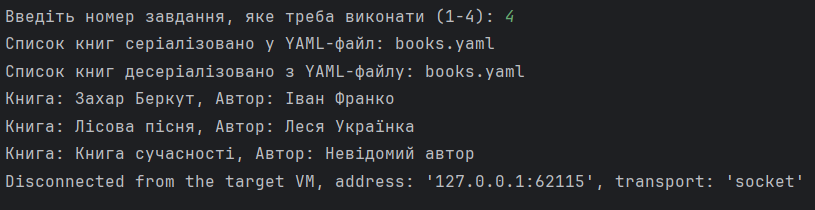


Рис. 3. Виконання YamlSerialisation

**ВИСНОВКИ**

У ході виконання лабораторної роботи я навчився працювати з потоками введення-виведення (I/O Streams), текстовою інформацією, файлами, нативною серіалізацією та сторонніми бібліотеками для серіалізації об'єктів у формати JSON та YAML. Ознайомитися з використанням Maven для управління залежностями та збіркою проекту, вивчити особливості використання систем збірки проєктів.